Evolutionary Systems Applied to the Synthesis of a CPU Controller

Ricardo S. Zebulum[1,2] Marco Aurélio Pacheco[2,3] Marley Vellasco[2,3]

[1] CCNR, University of Sussex, Brighton, BN1 9SB UK,
e-mail:ricardoz@cogs.susx.ac.uk
[2] ICA - Pontificia Universidade Catolica do Rio de Janeiro - Brasil
[3] Depto de Engenharia de Sistemas e Computação, UERJ -RJ, Brasil

Abstract. Our work introduces an evolutionary approach applied to the design of digital circuits. Particularly, we address the case of synthesising a controller for a simple CPU, a case study which has not been tackled by other authors so far. We employ a novel circuit evaluation strategy that is able to cope with the problem of epistasis when using a gate level representation of the circuit; and new evolvable hardware systems paradigms derive from this new methodology. We show that the use of this new evaluation approach allows the achievement of smaller circuits and promises to be effective when the problem scales up. Furthermore, our methodology yields novel digital circuits comparing to conventional design.

Keywords: Evolutionary Hardware, Sequential Circuits, CPU control.

1 Introduction

This work applies artificial evolution as a tool for automatic synthesis of digital circuits. Many applications of evolutionary systems in the design of both digital[2] and analog circuits[3] have been presented recently. In the particular case of digital circuit evolution, most works have focused on the area of combinational circuits; the main problem is that CAD tools for combinational circuit design can easily outperform evolutionary systems when the later are applied without any heuristics.

Due to the reasons mentioned above, the authors decided to investigate the area of sequential circuits design through evolutionary systems, which seems to be more promising in terms of competitiveness with conventional CAD tools. Sequential circuits are those which use feedback by the use of devices called flip-flops [4]. We propose a new kind of evaluation methodology, in which internal points of the evolving digital circuits are assessed together with the circuit output. This has been a way devised by the authors to reduce the epistatic effect from the representation, which is further discussed in the paper. We selected a CPU controller to be evolved, since this illustrates a practical application for evolutionary systems.

This article is composed of five additional sections: section 2 briefly reviews the area of sequential systems design and conventional tools used for that purpose. Section 3 presents the target problem, i.e., the particular architecture of the CPU for which the control circuit will be designed. Section 4 shows our evolutionary approach and section 5 describes the evolved circuit. Finally, section 6 analyses our results.

2 Sequential System Design

Figure 1 illustrates the basic topology of a sequential circuit. It can be seen that a combinational circuit (formed by basic boolean gates) and storage elements are interconnected to form this kind of topology [4]. The sequential circuit receives binary information from its environment via the inputs. These inputs, together with the present state of the storage elements, determine the binary value of the outputs. A sequential circuit is specified by a time sequence of inputs, internal states and outputs [4].

SIS system is a state of art tool for synthesis and optimisation of sequential circuits [6]. One of the main features of this tool is the exploration of signal dependencies across the memory elements boundaries, instead of optimising logic only within the combinational blocks. SIS includes methods for state assignment, state minimisation, testing, retiming, technology mapping, verification, timing analysis and optimisation across register boundaries. However, the design specification must be supplied as a netlist of gates or a finite state machine transition table, which requires a prior knowledge of the system from the user. We will show that our evolutionary system does not need this kind of previous knowledge.

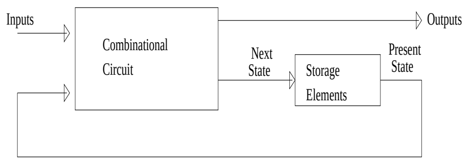


Fig. 1. Block Diagram of a Sequential Circuit (extracted from [4])

3 Target Problem - Random Control Logic Unit

The task of controlling the operations of a microprocessor is a typical example of a sequential circuit task. The control unit of a CPU consists of a program counter(PC), an instruction register(IR), and control logic; and may either be hardwired or microprogrammed. The control unit enables the CPU to carry out the instruction currently in the instruction register. In other words, there is a pattern of bits in the instruction register that is used to generate a sequence of actions taking place during the execution of an instruction [1]; the control unit is the circuit that provide this operation. Particularly, the random or hardwired logic control unit is made up of an arrangement of boolean gates.

In [1], a simple model of CPU is presented and a random logic control unit is designed for allowing the execution of eight different instructions. Using this CPU model we propose the task of evolving the control unit, instead of designing it. Figure 2 shows the structure of this primitive CPU; table 1 shows the interpretation of machine-code instructions (note that the fetch cycle occurs for all the eight instructions). Due to space reasons this table shows only some of the control signals, which are 16 in the total, being divided in clock signals(C), enable signals (E), flip-flop signals (Reset, Set), read and write. The evolutionary system must generate the signals C_{MAR}, E_{MBR}, E_{IR}, etc, given a particular instruction as input. For further details on the CPU operation, refer to [1].

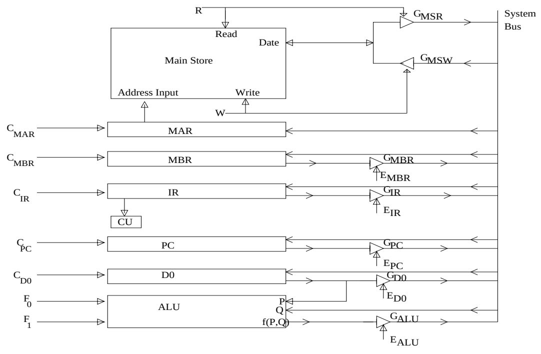


Fig. 2. Block Diagram of a Simple CPU (Reproduced from [1])

Instruction	Time	Enables					Clocks				Flip-Flop	
		MBR	IR	PC	D0	ALU	MAR	MBR	IR	PC	SET	RESET
Fetch	T0	0	0	1	0	0	1	0	0	0	0	0
	T1	0	0	0	0	0	0	0	1	0	0	0
	T2	0	0	1	0	0	0	0	0	0	0	0
	T3	0	0	0	0	1	0	0	0	1	0	0
	T4	0	1	0	0	0	1	0	0	0	1	0
Load	T0	0	0	0	0	0	0	0	0	0	0	1
Store	T0	0	0	0	1	0	0	0	0	0	0	1
Add	T0	0	0	0	0	0	0	1	0	0	0	0
	T1	1	0	0	0	0	0	0	0	0	0	0
	T2	0	0	0	0	1	0	0	0	0	0	1
Sub	T0	0	0	0	0	0	0	1	0	0	0	0
	T1	1	0	0	0	0	0	0	0	0	0	0
	T2	0	0	0	0	1	0	0	0	0	0	1
Inc	T0	0	0	0	0	0	0	1	0	0	0	0
	T1	1	0	0	0	0	0	0	0	0	0	0
	T2	0	0	0	0	1	0	1	0	0	0	0
	T3	1	0	0	0	0	0	0	0	0	0	1
Dec	T0	0	0	0	0	0	0	1	0	0	0	0
	T1	1	0	0	0	0	0	0	0	0	0	0
	T2	0	0	0	0	1	0	1	0	0	0	0
	T3	1	0	0	0	0	0	0	0	0	0	1
Bra	T0	0	1	0	0	0	0	0	0	1	0	1
Beq	T0	0	1	0	0	0	0	0	0	Z	0	1

{Table 1 - Interpretation of Machine Code Instructions (Reproduced from [1])

4 Problem Modelling

This section describes both the representation and evaluation used within our evolutionary system.

The gate level *representation* [2] has been used to encode each circuit into an integer string. Figure 3 illustrates an example of this kind of representation for a particular output signal. As there are a total of 16 output signals, the overall system will be made up of 16 cells like the one in the figure.

The genotype is made up of blocks of integer numbers or genes that encode each particular logic gate shown in the figure. The genes associated with gates of the first layer will encode its nature and also the source of the input signal. The cell input signals are chosen among the following signals (Figure 4):

1. Clock signals, supplied by a master-clock and a counter;
2. the three bits of the instruction register that determines the instruction to be executed;
3. and the own output control signals delayed by one clock period.

The delay flip-flop is a way devised by the authors to provide extra signals to be supplied to the cell's inputs: in this case, the own output signals delayed by one clock pulse. It has been verified that this procedure facilitates the task of the evolutionary system. The disadvantage of this approach is the increase in hardware requirements in terms of number of flip-flops. As it will be seen in the results' section, the authors sought for a compromise, by inserting the delay flip-flop in only some cells.

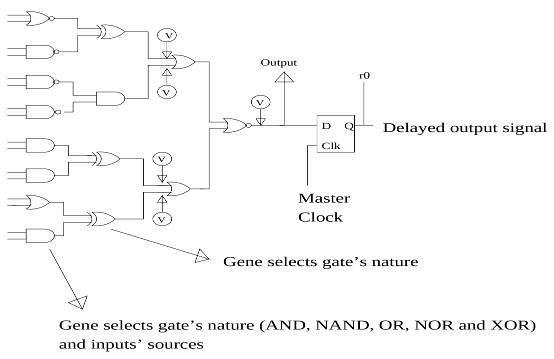

Output
r0
D Q
Clk
Delayed output signal
Master Clock
Gene selects gate's nature
Gene selects gate's nature (AND, NAND, OR, NOR and XOR) and inputs' sources
V

Fig. 3. Gate level represention of a sequential circuit

The *fitness evaluation function* was designed to simply count the number of hits in each cell output, comparing to the target output signals. The main problem of this evaluation function is the *epistasis* it introduces in the system. This fact can be visualised in Figure 5. Supposing that the boolean function

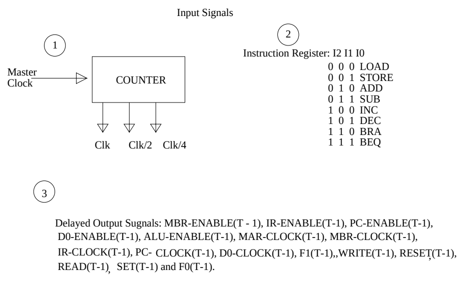

Input Signals
1
Master Clock
COUNTER
Clk
Clk/2
Clk/4
2
Instruction Register: I2 I1 I0
0 0 0 LOAD
0 0 1 STORE
0 1 0 ADD
0 1 1 SUB
1 0 0 INC
1 0 1 DEC
1 1 0 BRA
1 1 1 BEQ
3
Delayed Output Sugnals: MBR-ENABLE(T - 1), IR-ENABLE(T-1), PC-ENABLE(T-1), D0-ENABLE(T-1), ALU-ENABLE(T-1), MAR-CLOCK(T-1), MBR-CLOCK(T-1), IR-CLOCK(T-1), PC- CLOCK(T-1), D0-CLOCK(T-1), F1(T-1),,WRITE(T-1), RESET(T-1), READ(T-1), SET(T-1) and F0(T-1).

Fig. 4. Inputs available for the evolutionary system

$a.\bar{b} + \bar{a}.c$ is to be evolved, it can be seen that circuit of Figure 5.a is evaluated only by its output value, F_1, which is always zero in the case. Therefore, the circuit scores just four hits out of eight (see truth table in the Figure), though a simple mutation, changing the output gate from an AND to an OR, would solve the problem. This illustrates the fact that close genotypes may be not close in fitness. In Figure 5.b, instead, we fix the output gate to an OR gate, and the fitness is given by $F_1 + F_2 + F_3$. This methodology brings the following advantages:

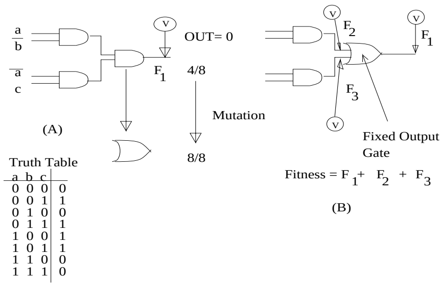
1. As internal points of the circuit are being probed, useful sub-circuits can be identified;
2. As the fitness is now a sum of terms, small changes in the genotype will have less impact in the overall fitness, diminishing then epistatic effects.

The output gates can either be fixed to the OR or NOR functions, corresponding respectively to the OR and NOR evolvable hardware paradigms. In the former, the target function for the internal points will be the same of the output point; in the later, it will be the complemented value of the target function. This is due to the fact that OR and NOR gates perform boolean sum and complemented boolean sum respectively. Fixing the output gate to other kind of logic function will increase the complexity of the evaluation function.

Additionally, when the output gate is fixed as an OR function, there will be heavy penalties when internal circuit points produces logic '1' value when the target is a logic '0' value, since this '1' value will also clamp the output to an erroneous value, regardless of the values of other internal points.

In the circuit of Figure 3, the output has been fixed to a NOR boolean function (2 ORs followed by 1 NOR = 1 NOR); five points are then evaluated, the final output and four internal points.

As a more quantitative measure of performance improvement using this methodology, a correct circuit to generate the WRITE signal has been achieved in 9 out of 10 trials using the NOR paradigm, whilst no successful execution have been obtained without using this paradigm.



a	b	c	
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Fig. 5. New evaluation methodology

5 Results

In order to evolve the whole control system, 16 genetic algorithms have to be executed, one for each circuit output. It has been verified that unsuccessful results are obtained when using one genetic algorithm to evolve more than one output. The authors adopted the following strategy:

1. Run the 16 GAs for each signal, assuming a delay flip-flop in each circuit output;
2. Find the output signal(s) which was(were) hardest to evolve, and store the delayed output signals used as inputs to this(these) cell(s);
3. Re-run the GA for the other signals, keeping available only the delayed signals used by the circuit representing the signals mentioned in the second item;

As stated previously, the idea of placing a flip-flop delay in the output of each cell is a means to give more resources to the genetic algorithm to find a solution. Nevertheless, in order to minimise the increase in hardware yielded by this approach, the method described above is used, i.e., placing flip-flop delay only in those signals used by the cells which have been hardest to evolve.

In our particular case, the RESET signal has been the hardest to be evolved. Its circuit is shown in Figure 6. From this Figure, it can be seen that the OR paradigm has been used for the evolutionary process. In the final evolved circuit, it has been verified that the cell could be simplified by taking away a sub-circuit which was not effectively contributing to the final behaviour. The possibility of cutting hardware from the final solution is another advantage of the OR paradigm.

In order to evolve the other signals, we allowed the GA to use only the delayed output signals used in the RESET circuit, which are shown in the Figure. After simplifying the circuit, it can be verified that only 5 out of 16 delayed output signals will be used, meaning that 5 additional flip-flops are going to be required.

Due to space reasons, we will not show the overall circuit. For some signals the OR-paradigm proved to be more efficient, while for others the NOR paradigm

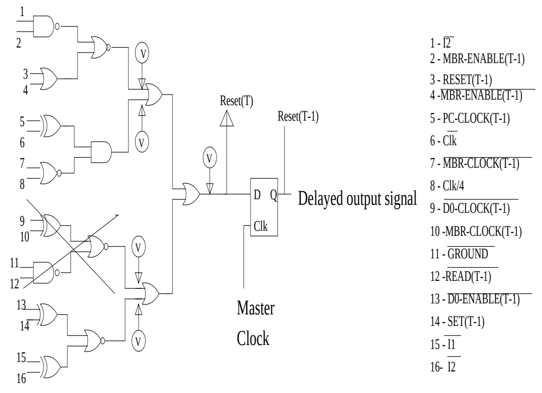

Reset(T)
Reset(T-1)
D
Q
Clk
Delayed output signal
Master
Clock
V
1 - $\overline{I2}$
2 - MBR-ENABLE(T-1)
3 - RESET(T-1)
4 - $\overline{\text{MBR-ENABLE(T-1)}}$
5 - PC-CLOCK(T-1)
6 - $\overline{\text{Clk}}$
7 - $\overline{\text{MBR-CLOCK(T-1)}}$
8 - Clk/4
9 - $\overline{\text{D0-CLOCK(T-1)}}$
10 - MBR-CLOCK(T-1)
11 - $\overline{\text{GROUND}}$
12 - $\overline{\text{READ(T-1)}}$
13 - $\overline{\text{D0-ENABLE(T-1)}}$
14 - SET(T-1)
15 - $\overline{I1}$
16 - $\overline{I2}$

Fig. 6. Circuit evolved to generate the RESET control signal

yielded better results. The graph of Figure 7 compares the evolution of the particular signal ALU-ENABLE when the OR and NOR paradigms are used. The average value of the best genotypes over five executions, along 300 generations for 40 individuals is shown in this graph. When using the OR paradigm, 4 out of 5 executions were successful, while 5 out of 5 were successful when using the NOR paradigm. It took around 4 minutes to run the executions in a SPARC 4 workstation.

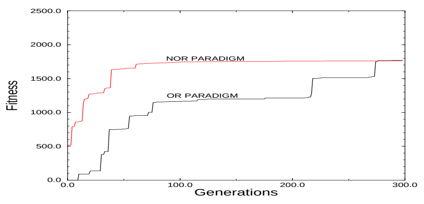


Fig. 7. Average Fitness of the Best Genotypes for the ALU-ENABLE signal using OR and NOR Paradigms

One of the evolved circuits for the ALU-ENABLE signal is shown in Figure 8. It can be verified that this circuit could be utterly simplified, and there is no need for an output flip-flop.

6 Analysis of the Results

We can compare the evolved CPU controller with a human designed one shown in [1]. The evolved circuit uses five additional flip-flops, meaning that evolutionary

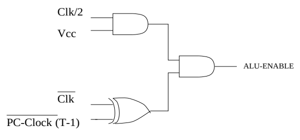


Fig. 8. Circuit evolved to generate the ALU-ENABLE control signal

systems does not use the minimum amount of states in the synthesis of the sequential system. In terms of boolean gates, the evolved controller uses around 150 gates, against 90 of the human designed one. Nevertheless, it seems possible to evolve controllers with around 100 boolean gates.

The increase in hardware requirements to synthesise the controller is somewhat expected, as evolutionary systems usually does not come to parsimonious solutions for design problems [3]. However, the proposed system has the advantage of using minimum designer knowledge and arriving at novel digital circuits. The former property reveals an advantage over conventional CAD tools like SIS; the latter property can be promising for more complex designs.

The authors have also presented a new evolvable hardware system methodology, in which internal circuit points are evaluated; this methodology can overcome epistasis problems and it proved to be essential in the synthesis of some control signals.

7 Acknowledges

The authors wish to thank CAPES, Brazilian federal agency, for the support.

References

1. Clements, A., "The Principles of Computer Hardware", Oxford University Press, 1991
2. Higuchi, T., Iba, H., Manderick, B., "Evolvable Hardware", in Massively Parallel Artificial Intelligence (ed. H. Kitano), MIT Press, 1994.
3. Koza J. R., Bennett III F. H., Andre, D. , Keane, M. A., "Four Problems for which a Computer Problem Evolved By Genetic Programming is Competitive with Human Performance", Proc. Of ICEC-96, IEEE Press., Pages 1-10.
4. Morris, M., Kime, C. R., "Logic and Computer Design Fundamentals", Prentice-Hall International Inc., 1997.
5. Muller, J. F., Thomson, P. , "Combinational and Sequential Logic Optimization Using Genetic Algorithms", Proc. of the First IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems , pp.34-38, UK, 1995.
6. Sentovich, E. M., Singh, K. J., Moon, C., Savoj, H., Brayton, R.K., Sangiovanni-Vincentelli, A., "Sequential Circuit Design Using Synthesis and Optimization", Proceedings of the IEEE Int. Conf. on Computer Design, pp. 328-333, 1992.
7. Zebulum, R. S., Pacheco, M. A., Vellasco, M., "Comparison of Different Evolutionary Methodologies Applied to Electronic Filter Design", Proc. of IEEE International Conference on Evolutionary Computation, Alaska, May, 1998.